

Systèmes d'Exploitation Centralisés Réalisation d'un mini-shell

Master 1re année

Spécialité Informatique

1 But du projet

Réaliser un mini-shell de nom mysh capable d'exécuter des programmes. L'ensemble des fonctionnalités de ce mini-shell est décrite dans les paragraphes suivants.

2 Lancement de commandes

La première fonctionalité est de pouvoir exécuter des programmes lus depuis une invite de commandes indiquant le répertoire courant (~ représentera la *home directory* de l'utilisateur). Votre programme attendra la fin de l'exécution de la commande avant de réafficher son invite de commandes.

2.1 Séquencement

Les commandes pourront être enchainées inconditionnellement par des ;.

Exemple 1

```
~> ls; cat /etc/passwd
```

Elles pourront également être enchainées conditionnellement avec les opérateurs && et ||. La commande suivant un && (réciproquement ||) n'est exécutée que si et seulement si la commande le précédant a réussi (réciproquement échouée).

Exemple 2

```
~> gcc -o mysh myshell.c && ./mysh
~> test -d .can || mkdir .can
```

Dans l'exemple précédent, l'exécution de la commande mysh ne sera faite que si la compilation de myshell. c s'est achevée correctement. Inversement la ligne suivante créera le répertoire .can si celui n'existe pas, c'est-à-dire si la commande test a échoué.

2.2 Wildcards

Les lignes de commandes de votre shell devront remplacer les caractères « jokers » par leur(s) correspondance(s) calculée(s) à partir du répertoire courant. Les caractères à gérer sont :

- * : qui se remplace par une suite quelconque de caractère. Cette suite de caractères peut éventuellement être vide ;
- ? : qui se remplace par exactement un caractère ;
- [ens]: qui correspond exactement à un caractère de l'ensemble de caractères définis par ens. Cet ensemble peut être défini soit par énumération de caractères, soit par ntervalle, soit par combinaison d'énumération et d'intervalle. Lorsqu'un ensemble est précédé du caractère ^ les caractères de cet ensemble sont interdits au lieu d'être autorisés pour la correspondance.

Exemple 3

```
~> cat *.[ch]
~> ls ../[A-Z.]*[^~]
~> wc -l /etc/?????
```

Dans l'exemple précédent, la première ligne de commandes permet de visualier le contenu de tous les fichiers se terminant par l'extension . c ou . h. La deuxième ligne permet de lister toutes les entrées du répertoire père qui commencent par une majuscule ou par un point et qui ne se terminent pas par ~. La dernière ligne permet de compter les lignes de tous les fichiers dont le nom comporte exactement 5 caractères et qui se situent dans /etc.

Il est à noter que les caractères « jokers » précédés d'un \ ne sont pas à remplacer.

3 Commandes

3.1 Changement de répertoire

Afin de pouvoir se ballader dans l'arborescence de fichiers, voter shell devra disposer d'une commande interne cd permettant de se déplacer dans le répertoire passé en paramètre de cette commande. Si aucun répertoire n'est donné à cette commande, le déplacement se fera vers la *home directory* de l'utilisateur courant du shell.

3.2 Sortie du Shell (et propagation du Ctrl-C)

Deux moyens seront offerts pour quitter votre shell. La commande interne exit qui quittera votre shell sans tuer les commandes lancées en tâche de fond (*background*, voir paragraphe 5) et le Ctrl-C qui demandera une confirmation avant de quitter mais qui tuera tous les processus en tâche de fond.

Il est à noter que si un Ctrl-C survient alors qu'il y a une commande en cours d'exécution (en *foreground*), ce Ctrl-C devra alors être propagé au processus en cours d'exécution et ne sera pas considéré comme une demande de sortie de votre shell.

3.3 Code de retour d'un processus

Votre shell intégrera la commande interne status qui affiche pour le dernier processus (xxxx) exécuté en premier plan (en *foreground*) :

- xxxx terminé avec comme code de retour YYY si le processus s'est terminé normalement;
- xxxx terminé anormalement dans le cas d'une terminaison anormale (comme par exemple l'interruption via un Ctrl-C).

3.4 Lister le contenu d'un répertoire

En programme externe à votre shell, il vous est demandé également de programmer une commande de nom myls qui sera équivalente à la commande ls -l et qui acceptera deux options -a et -R. Ces options permettent respectivement de lister les fichiers cachés et d'explorer récursivement le ou les répertoires passés en paramètre de la commande. Votre myls sera par défaut en couleur, une couleur sera attribuée à chaque type de fichier. Les options peuvent être combinées. Si aucune entrée à lister n'est passée en paramètre, par défaut le répertoire courant est lister. Voici quelques exemples de commandes valides qui devront être gérées :

Exemple 4

3.5 Afficher l'état des processus en cours

Également en programme externe, vous devez réaliser la commande myps équivalente à la commande ps aux à la différence que votre programme affichera en couleur les processus en fonction de leur état.

4 Les redirections

4.1 Les tubes

Vos lignes de commandes devront permettre le « pipeline » de commandes au travers du symbole |. Ce symbole permet de rediriger la sortie standard du programme précédant le | vers l'entrée standard du programme suivant le |.

Les commandes pourront être « pipelinées » à l'infini.

Exemple 5

```
~> ls | sort -r
~> ps | grep mysh | wc -l
```

Le premier exemple permet de trier dans l'ordre alphanumérique inverse les entrées du répertoires courant, alors que le second permet de compter le nombre de processus mysh en cours d'exécution.

4.2 Redirections vers ou depuis un fichier

Vous devrez gérer les redirections usuelles vers ou depuis les fichiers à savoir :

- > redirige la sortie standard de la commande vers le fichier. Le précédent contenu du fichier est perdu ;
- >> redirige la sortie standard de la commande vers le fichier. L'écriture s'effectue en fin de fichier, le précédent contenu du fichier est donc conservé;
- 2> idem que > pour la sortie standard des erreurs ;
- 2>> idem que >> pour la sortie standard des erreurs ;
- >& redirige la sortie standard et la sortie standard des erreurs de la commande vers le fichier. Le précédent contenu du fichier est perdu;
- >>& redirige la sortie standard et la sortie standard des erreurs de la commande vers le fichier. L'écriture s'effectue en fin de fichier, le précédent contenu du fichier est donc conservé;
- < redirige l'entrée standard de la commande vers l

Il est à noter que, quelle que soit la redirection utilisée, la syntaxe est toujours de la forme : $commande\ Redirection\ fichier$

Exemple 6

La première commande ajoute au fichier listofsongs de /home/ens/mazure les fichiers dont l'extension est .mp3 du répertoire courant. La seconde commande affiche les lignes numérotées du fichier myshell.c. La toisième commande récupère toutes les fichiers d'inclusion (exceptés ceux des librairies standard) du fichier myshell.c.

5 Premier et arrière plans

Jusqu'à présent toutes les commandes exécutées, l'étaient en premier plan ou en « foreground », c'està-dire que l'on attendait la fin de l'exécution de l'ensemble de la ligne de commandes avant d'afficher de nouveau l'invite de commandes. Votre shell devra prévoir le lancement de commandes en arrière-plan ou en

« background », c'est-à-dire que l'invite de commandes sera affichée sans attendre la fin de la commande. Pour celà un & suivra la commande à lancer en background.

Exemple 7

```
~> emacs & 
~> ls -lR | gzip > ls-lR.gz &
```

Au lancement d'une commande en arrière plan, avant de réafficher le l'invite de commandes, votre shell affichera une ligne de la forme : [xxx] yyy où xxx représente le numéro du « job » en « background » et yyy le pid du processus. Le numéro de job est un compteur qui est réinitialisé lorsque plus aucun job n'est exécuté en arrière plan.

Lorsqu'une commande lancée en arrière plan se termine, on affichera : zzz (jobs=[xxx], pid=yyy) terminée avec status=sss où zzz représente la commande lancée, xxx le numéro de job, yyy le pid et sss le code de retour de la commande (la valeur -1 sera affichée si la commande s'est terminée anormalement).

5.1 Commande myjobs

La commande interne myjobs permettra d'afficher la liste des processus en arrière plan. L'affichage se fera de la manière suivante : [xxx] yyy Etat zzz où xxx représente le numéro de job, yyy le pid, Etat l'état du processus qui pourra être En cours d'exécution ou Stoppé et zzz la commande lancée. Un job par ligne sera affiché.

5.2 Passer une commande de foreground à background et inversement

Lorsqu'une commande est lancée en *foreground*, il est possible de l'interrompre en lui propageant le signal envoyé par Ctrl-Z (signal SIGTSTP). La commande est alors stoppée, votre shell reprend la main en indiquant que la commande zzz devient le job xxx et qu'il est Stoppé.

Les commandes internes myfg et mybg permettent de modifier l'état d'un job. Ces commandes peuvent admettre un numéro de job en paramètre. La commande mybg permettra de passer le job stoppé ou en foreground de plus grand numéro ou de numéro passé en paramètre en exécution en arrière plan. Inversement, la commande myfg permettra de passer le job stoppé ou en background de plus grand numéro ou de numéro passé en paramètre en exécution en premier plan. Le signal SIGCONT permet la demande de reprise d'un processus précédemment stoppé. Si la commande mybg s'applique à un numéro de processus déjà en « background », un erreur est signalée et aucune changement n'est réalisé sur les jobs.

6 Les variables

Pour être un interpréteur de commandes qui se respecte, votre shell doit intégrer la prise en charge de variables. Deux types de variables sont à gérer : les variables locales et d'environnement.

6.1 Les variables d'environnment

Les variables d'environnement sont stockées dans une zone de mémoire partagée qui est créée par le premier exemplaire de votre shell qui tourne à un instant donné et est initialisée avec toutes les variables définies dans le tableau envp (int main(int argc,char *argv[],char *envp[])). Le dernier exemplaire de votre shell qui tourne à un instant donné doit détruire cet espace partagé avant de mourir.

Cette zone de mémoire partagée est à gérer comme une mémoire par subdivison.

La zone de mémoire partagée est une zone de mémoire à accès concurrents avec plusieurs lectures simultanées possibles mais 1 seule écriture possible à un instant donné. L'implémentation demandée doit donner la priorité à l'écriture.

6.2 Les variables locales

Elles sont locales à votre shell, c'est-à-dire que dex exemplaires de votre shell peuvent avoir un même nom de variable avec une valeur différente dans chaque shell.

6.3 Utilisation des variables

Pour accéder à une variable sur la une ligne de commande, on utilisera le symbole \$ devant son nom.

Si une variable de nom nnn existe à la fois en locale en environnement, la variable locale est prioritaire. La valeur de cette dernière sera donc utilisée.

Pour définir la valeur d'une variable on utilisera respectivement les commandes internes set et setenv pour les variables locales et d'environnment. Ces commandes admettent un paramètre de la forme : var=valeur où var représente la variable à définir et valeur la valeur à lui associer. Si ces commandes sont lancées sans paramètre, elles afficheront les respectivement les valeurs de l'ensemble des variables locales et d'environnment.

Les commandes internes unset et unsetenv admettent obligatoirement un paramètre qui est le nom de la variable à supprimer et ont pour objectif de supprimer respectivement une variable locale ou d'environnement.

Exemple 8

```
~>set a=foo
~>echo $a
foo
~>setenv b=duc
\sim>mysh
~>echo $b
duc
~>set a=bar
\sim>set b=tmp
~>echo $b
tmp
~>setenv b=$a
\sim>exit
~>echo $b+$a
bar+foo
~>unset $a
```

7 Modalité de remise du projet

Voir avec vos enseignants	de TP.		
Bon courage!			